1b

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|---|---|---|---|---|
| 10/779,503 | 02/13/2004 | David Arthur James Webb JR. | 200304950-1 | 9855 |

7590 09/22/2005

Hewlett-Packard Company
Intellectual Property Administration
P.O. Box 272400
Fort Collins, CO. 80527-2400

| EXAMINER | |
|---|---|
| HUISMAN, DAVID J | |
| **ART UNIT** | **PAPER NUMBER** |
| 2183 | |

DATE MAILED: 09/22/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

K

| Office Action Summary | Application No. 10/779,503 | Applicant(s) JAMES WEBB ET AL. | |
|---|---|---|---|
| | Examiner David J. Huisman | Art Unit 2183 | |

*-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --*

**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

1)☒ Responsive to communication(s) filed on *30 June 2005*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

4)☒ Claim(s) *1-4,7,8,10,12-14,17,18 and 20-31* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-4,7,8,10,12-14,17,18 and 20-31* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

**Application Papers**

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *13 February 2004* is/are: a)☐ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. § 119**

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

1) ☐ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application (PTO-152)

6) ☐ Other: _____.

[illegible]

## DETAILED ACTION

1. Claims 1-4, 7-8, 10, 12-14, 17-18, and 20-31 have been examined.

### *Papers Submitted*

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: Amendment and Terminal Disclaimer as received on 6/30/2005.

### *Terminal Disclaimer*

3. The terminal disclaimer filed on June 30, 2005 disclaiming the terminal portion of any patent granted on this application which would extend beyond the expiration date of U.S. Patent No. 6,738,896 has been reviewed and is accepted. The terminal disclaimer has been recorded.

### *Drawings*

4. The drawings are objected to under 37 CFR 1.83(a). The drawings must show every feature of the invention specified in the claims. Therefore, the limitation from claims 8 and 25-28 must be shown or the feature(s) canceled from the claim(s). No new matter should be entered.

Corrected drawing sheets in compliance with 37 CFR 1.121(d) are required in reply to the Office action to avoid abandonment of the application. Any amended replacement drawing sheet should include all of the figures appearing on the immediate prior version of the sheet, even if only one figure is being amended. The figure or figure number of an amended drawing should not be labeled as "amended." If a drawing figure is to be canceled, the appropriate figure must be removed from the replacement sheet, and where necessary, the remaining figures must be renumbered and appropriate changes made to the brief description of the several views of the drawings for consistency. Additional replacement sheets may be necessary to show the renumbering of the remaining figures. Each drawing sheet submitted after the filing date of an application must be labeled in the top margin as either "Replacement Sheet" or "New Sheet" pursuant to 37 CFR 1.121(d). If the changes are not accepted by the examiner, the applicant will be notified and informed of any required corrective action in the next Office action. The objection to the drawings will not be held in abeyance.

## *Maintained Rejections*

5. Applicant has failed to overcome the prior art rejections set forth in the previous Office Action. Consequently, these rejections are respectfully maintained by the examiner and are copied below for applicant's convenience.

## *Claim Rejections - 35 USC § 112*

6. The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

7. Claims 7-8 and 17-18 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the enablement requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to enable one skilled in the art to which it pertains, or with which it is most nearly connected, to make and/or use the invention. Applicant has

amended claims 7 and 17 to state that the divisor of the modulus operation is equal to the queue entry number, which corresponds to a particular queue location. Applicant has not enabled such an operation, and it is not clear how such an operation would provide any benefit. For instance, applicant's specification states, on page 7, lines 22-24, that the divisor is equal to the total number of locations in the queue, which is what applicant had previously claimed, and which makes sense to the examiner. As an example, if a queue had 8 entries (0-7), then the divisor would be 8. This way, a load instruction having an ID would map to any location in the queue by performing a "modulo 8" operation. The examiner does not understand how the current claim would work properly. For purposes of this examination, the claim will be interpreted as if the divisor equals the total number of locations in the queue.

## *Claim Rejections - 35 USC § 103*

8. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

9. Claims 1-4, 7-8, 10, 12-14, 17-18, and 20-31 are rejected under 35 U.S.C. 103(a) as being unpatentable over Akkary et al, U.S. Patent Number 6,182,210 (herein referred to as Akkary), in view of Popescu et al. (IEEE Micro, Vol. 11 Issue 3, Jun 1991).

10. Referring to claims 1 and 10 Akkary has taught a method for executing program instructions comprising the steps of:

a) assigning each individual ones of a plurality of program steps a unique number (Akkary figure 2 and 24 column 9 lines 6-10, column 16 lines 42-49, column 18 lines 57-58; the reason for assigning IDs to the instruction is for maintaining a record of the original program order so you can piece things back together properly later. Also, if the usage of the ID's is to determine age and/or to piece things back together after out of order execution, then they would need to be in an order and unique, otherwise you couldn't determine the original order unless there was some pattern to the assignment; the same reasoning is used for assigning load IDs).,

b) issuing a program step to an execution queue out of program order (Akkary figure 2 and 24 column 16 lines 42-49; some would go from the sched/issue unit 156 to the exec units 158 and some to the load or store buffers, 182 and 184; based on Applicant's definition of random, see specification page 9 lines 6- 11). In addition, note that column 4, lines 38-39, disclose that the processor is an out-of-order processor.

c) selecting a specific one of a plurality of locations in the execution queue based upon the unique number of the issued program step, each location having an instruction valid bit and the execution queue having a certain total number of locations (Akkary figure 2 and 24 column 18 line 54-column 19 line 23, column 24 lines 25-34; the load queue would have to have a pointer or counter, or both to keep track of which spot is the next location in the queue that the next sequential load instruction will be stored, therefore the next sequential instruction ID, or load buffer ID, will be selected and then selecting the specific location in the queue that is the next location that will be available when a load instruction is retired in program order);

d) determining the value of the instruction valid bit associated with the selected location (Akkary figure 2 and 24 column 18 line 54-column 19 line 23, column 24 lines 25-34; inherently the load queue would have to check the valid bit before it stored the next load instruction in the location, otherwise it would overwrite the current load instruction when it is still valid);

e) determining availability of the selected numbered location such that the issued program step is stored in the selected numbered location if the selected numbered location is available (Akkary figure 2 and 24 column 18 lines 58-62), and issuing an indication that the execution queue is full if the selected location is not available (Akkary figure 2 and 24 column 19 lines 14-16 and lines 31-37; the load queue must send some indicator to the rename/allocate unit 150 if it cannot accept anymore load instructions, otherwise the system would not work properly since the rename/allocate unit 150 would continue to send instructions and would overwrite instructions that have not yet been retired; this would be based on the comparing the valid entry bits).

Akkary has not taught wherein:

based on the unique number of the issued program step, calculating value of a status bit for the selected location; and

based upon the determined value of the instruction valid bit and the calculated value of the status bit, determining availability of the selected location.

f) However, Akkary has taught using a valid entry field to determine if the location in the queue is a valid instruction, and if the queue is full not allowing more instructions to be allocated to the queue (Akkary figure 2 and 24 column 19 lines 14-16 and lines 31-37). Akkary has also taught using a head pointer and tail pointer in the queue (Akkary figure 2 and 24 column 24 lines 25-29).

Popescu et al. has taught wherein calculating value of a status bit, or a color bit, for the selected location, using the certain total number and the unique number of the issued program step and based upon the determined value of the instruction value bit and the calculated value of the status bit, determining availability of the selected location. Popescu et al. taught that the index position of the instruction in a queue wraps around from its maximum value back to its minimum value, and using a 'color' bit to determine where the older instructions start by flipping the color bit when the index of the queue wraps around (Popescu et al. page 64 paragraph 4; In order for an index to wrap around, there must inherently be a modulus operation of the location number occurring, otherwise if you have a 16 entry queue, and you get to the 17th instruction, and the last entry is tilled, you would have to wait for the last entry to become available, in this manner, allowing the queue to wrap around to the top, instructions can continually be sent to the load queue as the oldest instructions, or locations, retire and become available - also, the total number of queue entries would have to be includes since the modulus calculation would be based on the total number of locations available). A numeric calculation would have to be done to see which location the next instruction will be selected to go. This calculation would include a modulus type function since the queue wraps around to the beginning once it reaches the end. The color bit would be the status bit, and the valid entry field of Akkary would still need to be checked to see if that specific location in the queue, or buffer, is available. It would have been obvious to one of ordinary skill in the art at the time of the invention to use a wrap-around queue as taught by Popescu et al. to allow instructions to continually be issued to the load queue as resources and/or locations become available. This will increase the throughput of the load queue, and thus the throughput for all the instructions of the program since some instructions will be dependent on the load instructions. Having a queue that wraps around will also prevent the queue from 'shifting' the entries in the location up to the beginning every time an instruction is retired. This will reduce the amount of hardware required for the unit. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to select a location by calculating and recording a modulus of the location number to increase throughput of the system and to reduce the hardware needed in the system.

11. Referring to claims 2 and 12 Akkary has not taught the method of claim 1, wherein the unique numbers are monotonically ascending series of integers assigned to the program steps in the same order as the program is executed (Akkary column 9 lines 6-10, column 16 lines 42-49, column 1 8 lines 57-58). However, the use of a monotonically increasing integer to be a unique number for each program step would have been obvious. A strong motivation at the time of invention for using a monotonically increasing integer is to simplify the logic necessary to create the unique number. One of ordinary skill in the art at the time of invention would have known that one could simply use an integer counter to generate the unique ID at the time the instructions are issued and would have been motivated to do so to reduce chip cost.

12. Referring to claims 3 and 13 Akkary has taught a method as described in claim 1, wherein the execution queue is one of a load queue and a store queue (Akkary figure 2 reference numbers 182 and 184).

13. Referring to claims 4 and 14, Akkary has taught the method of claim 1, wherein the program step is issued to the execution queue in an order determined by an availability of a selected one of a plurality of computation resources, the determined order different than the program order (Akkary figure 2 column 18 line 54-column 19 line 8, column 24 lines 25-34; the load instructions are issued to the load queue when resources and locations are available, after a load has retired and been deallocated using the valid and/or status bits).

14. Referring to claims 7 and 17 Akkary has taught the method of claim 21, wherein the divisor of the modulus operation is equal to the queue entry number (Popescu et al. page 64 paragraph 44 Inherently for the modulus operation to allow the location numbers to wrap around the queue, the divisor of the modulus must be the number of locations, otherwise, when the modulus is used to decide which location to put an instruction it may not calculate a location that exists in the queue).

15. Referring to claims 8 and 18, Akkary has taught the method of claim 7, further comprising switching the status bit of the selected location when the selected location becomes invalid (Akkary column 19 lines 14-16; the bit would have to be switched from valid to invalid when it becomes available, otherwise no new load instructions could be issued to the queue because all of the locations would be marked valid).

16. Referring to claim 20, Akkary in view of Popescu has taught a method as described in claim 1. Akkary in view of Popescu have further taught performing a numeric operation on the unique number of the issued program step to calculate a queue entry number that selects the specific one of the plurality of locations in the execution queue. In essence, since a wrap-around queue is implemented, then a modulus operation must be performed on the LBID so that the same X entries will be accessed over time. For instance, if the queue had 8 entries, the operation would be LBID % 8 (where % denotes a modulus operation). With this operation, the 1st load (LBID = 0) would be assigned to the 1st entry (entry 0), the 2nd load would be assigned to entry 1, and the 9th load (LBID = 8) would be assigned to the 1st entry (entry 0). This is how wrap-around is achieved.

17. Referring to claim 21, Akkary in view of Popescu has taught a method as described in claim 20. Akkary in view of Popescu have further taught that performing the numeric operation comprises performing a modulus operation on the unique number to calculate the queue entry number that selects the specific one of the plurality of locations in the execution queue. This is rejected for the same reasons set forth in the rejection of claim 20.

18. Referring to claim 22, Akkary in view of Popescu has taught a method as described in claim 21. Akkary in view of Popescu have further taught that performing the modulus operation also produces the value of the status bit. The status bit (color bit) is toggled when wrap-around occurs. Wrap-around occurs when a remainder of a modulus operation equals 0. So, if a queue holds 8 instructions, then on the 9th instruction, the modulus operation would yield a remainder of 0 (8 % 8 = 0). This will again map to the first entry and so wrap-around has occurred. Consequently, the status bit must be set.

19. Referring to claim 23, Akkary in view of Popescu has taught a method as described in claim 20. Akkary in view of Popescu have further taught that performing the numeric operation also produces the value of the status bit. This is rejected for the same reasons set forth in the rejection of claim 22.

20. Referring to claim 24, Akkary in view of Popescu has taught a method as described in claim 20. Akkary in view of Popescu has further taught that each location of the execution queue further is associated with a respective stored status bit, and wherein determining availability of the selected location is based on the determined value of the instruction valid bit and on comparing the calculated value of the status bit with the stored status bit associated with the selected location. As described above, each entry has a determined valid bit. Then when

another instruction maps to one of the entries, the calculated status bit for that instruction is compared to the determined status bit of the entry. If they match, the queue is empty; if they differ, the queue is full.

21. Referring to claim 25, Akkary in view of Popescu has taught a method as described in claim 24. Akkary in view of Popescu has further taught that the selected location is determined to be unavailable in response to determining that the stored status bit associated with the selected location does not match the calculated status bit. Again, if the determined status bit does not match the calculated status bit, then the queue is full, and so the entry is unavailable.

22. Referring to claim 26, Akkary in view of Popescu has taught a method as described in claim 25. Akkary in view of Popescu has further taught changing a state of the stored status bit of each of the plurality of locations of the execution queue in response to completion of all program steps stored in the plurality of locations. If the queue is full and another instruction is waiting to be entered (status bit matches with an instruction already in the queue), then when the instruction in the queue completes, the only way to signal that the entry is free is to change the status bit so that the status bits do not match.

23. Referring to claim 27, Akkary in view of Popescu has taught an apparatus as described in claim 10. Furthermore, the apparatus of claim 27 performs the method of claim 21. Consequently, claim 27 is rejected for the same reasons set forth in the rejection of claim 21.

24. Referring to claim 28, Akkary in view of Popescu has taught an apparatus as described in claim 27. Furthermore, the apparatus of claim 28 performs the method of claim 22. Consequently, claim 28 is rejected for the same reasons set forth in the rejection of claim 22.

25. Referring to claim 29, Akkary in view of Popescu has taught an apparatus as described in claim 28. Furthermore, the apparatus of claim 29 performs the method of claim 24. Consequently, claim 29 is rejected for the same reasons set forth in the rejection of claim 24.

26. Referring to claim 30, Akkary in view of Popescu has taught an apparatus as described in claim 29. Furthermore, the apparatus of claim 30 performs the method of claim 26. Consequently, claim 30 is rejected for the same reasons set forth in the rejection of claim 26.

27. Referring to claim 31, Akkary has taught an apparatus for executing program instructions comprising:

a) a first queue to store program steps in a program order, the program steps assigned respective program numbers that correspond to the program order. See Fig.2, component 104, for instance, and note that instructions are inherently stored in program order so that they may be sequentially fetched. They are fetched in order but issued out of order. It should further be noted that each load instruction in the program order is assigned an ID. See Fig.2 and 24, column 9, lines 6-10, column 16, lines 42-49, and column 18, lines 57-58.

b) an execution queue having a plurality of locations, the first queue to transfer at least some of the program steps to the execution queue out of the program order. See Fig.24 and note the load buffer for holding issued loads. In addition, from column 4, lines 38-39, note that the processor is an out-of-order processor.

c) Akkary has not taught that the locations of the execution queue to store respective program steps based on queue entry numbers calculated from numeric operations on the program numbers of respective program steps. However, Popescu has taught that the index position of the instruction in a queue wraps around from its maximum value back to its minimum value, and using a ‘color' bit to determine where the older instructions start by flipping the color bit when the index of the queue wraps around (Popescu et al. page 64 paragraph 4; In order for an index to wrap around, there must inherently be a modulus operation of the location number occurring, otherwise if you have a 16 entry queue, and you get to the 17th instruction, and the last entry is tilled, you would have to wait for the last entry to become available, in this manner, allowing the queue to wrap around to the top, instructions can continually be sent to the load queue as the oldest instructions, or locations, retire and become available - also, the total number of queue entries would have to be includes since the modulus calculation would be based on the total number of locations available). A numeric calculation would have to be done to see which location the next instruction will be selected to go. This calculation would include a modulus type function since the queue wraps around to the beginning once it reaches the end. The color bit would be the status bit, and the valid entry field of Akkary would still need to be checked to see if that specific location in the queue, or buffer, is available. It would have been obvious to one of ordinary skill in the art at the time of the invention to use a wrap-around queue as taught by Popescu et al. to allow instructions to continually be issued to the load queue as resources and/or locations become available. This will increase the throughput of the load queue, and thus the throughput for all the instructions of the program since some instructions will be dependent on the load instructions. Having a queue that wraps around will also prevent the queue from ‘shifting' the entries in the location up to the beginning every time an instruction is retired. This will reduce the amount of hardware required for the unit. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to select a location by calculating and recording a modulus of the location number to increase throughput of the system and to reduce the hardware needed in the system.

d) each location of the execution queue associated with an instruction valid bit for indicating whether the respective location is available (see Fig.24, "entry valid" bit), each location of the execution queue further associated with a stored status bit (Popescu's color bit), the first queue to further determine whether a particular location of the execution queue is available by:

d1) calculating a status bit based on the program number of the respective program step. Again, recall that Popescu teaches toggling the color bit when wrap-around occurs. In essence, since a wrap-around queue is implemented, a modulus operation must be performed on the LBID so that the same X entries will be accessed over time and so that wrap-around may be detected. For instance, if the queue had 8 entries, the operation would be LBID % 8 (where % denotes a modulus operation). With this operation, the 1st load (LBID = 0) would be assigned to the 1st entry (entry 0), the 2nd load would be assigned to entry 1, and the 9th load (LBID = 8) would be assigned to the 1st entry (entry 0) while toggling (calculating the status bit). This is how wrap-around is achieved and detected.

d2) comparing the calculated status bit with the stored status bit to determine whether the particular location is available. As described above, each entry has a determined valid bit. Then when another instruction maps to one of the entries, the calculated status bit for that instruction is compared to the determined status bit of the entry. If they match, the queue is empty; if they differ, the queue is full. If the queue is full, clearly, the particular location is unavailable.

## *Response to Arguments*

28. Applicant's arguments filed on June 30, 2005, have been fully considered but they are not persuasive.

29. Applicant argues the novelty/rejection of claim 1 on pages 9-10 of the remarks, in substance that:

> "Thus, selection of an entry of the buffer to place an instruction into, as taught by Akkary, is based on the pointer/counter referred to by the Office Action (the head and tail pointer/counter of Akkary), not "based upon the unique number of the issued program step" as recited in claim 1."

30. These arguments are not found persuasive for the following reasons:

a) Looking at Fig.24, it can be seen that each load instruction is assigned a load ID (LBID). The LBID is not only a unique number assigned to each load instruction, but it also corresponds to a particular entry in the load execution queue. It can be seen from Fig.24 that the 1st load instruction (Load 0) will go in the 1st entry of the queue, the 2nd load instruction (Load 1) will go in the 2nd entry of the queue, and so on. The LBID is essentially the counter/pointer which selects an entry in the queue.

31. Applicant argues the novelty/rejection of claim 1 on page 10 of the remarks, in substance that:

> "The color bit is not used to determine whether a selected location (selected based on a unique number of the issued program step) is available...The Office Action does not explain how the color bit of Popescu is used to determine whether the selected location is available, as expressly recited in claim 1. The wrap-around queue of Popescu does not use a status bit that is checked to determine availability of a selected location of the queue."

32. These arguments are not found persuasive for the following reasons:

a) As the examiner explained in the Office Action, the color bit allows for wrapping in a queue. A color bit (or wrap bit) is known in the art to assist in determining availability of a particular queue entry. For instance, as described in Popescu, as instructions are entered into a circular (wrap-around) queue, the queue will eventually become full with instructions having an associated color bit equal to 0. Once the queue becomes full, the next entry to receive an instruction would again be the very first entry, but this time the color bit would be toggled to 1 to indicate the occurrence of wrap-around. Therefore, the next instruction to be entered in the first entry would have a color bit of 1. Since the associated color bits of the instructions associated with the first entry are not equal, this indicates that the queue is full and that the next instruction cannot be entered into the queue. If the color bits are equal, then this indicates that the queue is empty, and that the entry would be available. So, the color bit does in fact determine availability in that there are no entries available if the queue if full.

33. Applicant argues the novelty/rejection of claim 1 on page 11 of the remarks, in substance that:

> "The teaching of Popescu with respect to shelved instructions, which are stalled instructions, stored in the DRIS is completely unrelated to the load buffer, used for storing load instructions, as taught by Akkary. A person of ordinary skill in the art looking to the disparate teachings of Akkary and Popescu would not have been motivated to combine the teachings regarding the DRIS in Popescu with the teachings regarding the load buffer of Akkary."

34. These arguments are not found persuasive for the following reasons:

a) The reason for combining the two references was to gain the advantages achieved from the use of a color bit. Even if Akkary's load buffer and Popescu's instruction shelf are different store different information, a person of ordinary skill in the art would've recognized that Popescu's color bit would be useful in Akkary in order to allow for queue wrap-around. Wrap-around saves time (and therefore increases throughput) and it also prevents having to ensure that the address pointer of the queue stay within the boundary of the queue.

## *Conclusion*

35. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David J. Huisman whose telephone number is (571) 272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

DJH
David J. Huisman
September 2, 2005

[illegible]

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100